

LAPORAN ANALISIS CIPHER KLASIK

Laporan ini disusun untuk memenuhi nilai Tugas Besar 1 Kriptografi

Dosen Pengampu : Kodrat Mahatma S.T.,M.Kom.



Disusun Oleh :

M. Taupik Anjana 20123005

Fitri Anisa 20123020

C1.23 INFORMATIKA

UNIVERSITAS TEKNOLOGI DIGITAL

2025

A. CIPHER KLASIK

1. CAESAR

1.1 Pengertian

Caesar Cipher merupakan salah satu bentuk cipher paling sederhana dan tertua dalam sejarah kriptografi. Algoritma ini ditemukan oleh Julius Caesar, seorang jenderal Romawi, untuk menyampaikan pesan militer secara rahasia kepada bawahannya.

Prinsip dasarnya adalah menggeser setiap huruf dalam plaintext sejauh n posisi dalam alfabet. Misalnya, jika pergeseran (kunci) = 3, maka huruf A akan berubah menjadi D, B menjadi E, dan seterusnya. Setelah huruf Z, proses geser akan kembali ke A.

Meskipun sederhana, Caesar Cipher menunjukkan konsep awal dari substitusi monoalfabetik, di mana satu huruf plaintext selalu diganti dengan satu huruf ciphertext yang tetap.

1.2 Rumus Matematis

$$E(x) = (x + k) \bmod 26$$

$$D(x) = (x - k) \bmod 26$$

Keterangan:

- x = posisi huruf plaintext ($A=0, B=1, \dots, Z=25$)
- k = jumlah pergeseran (kunci)
- $E(x)$ = ciphertext
- $D(x)$ = plaintext hasil dekripsi.

1.3 Implemetasi Python

```
def caesar_encrypt(text, shift):
    result = ""
    for char in text.upper():
        if char.isalpha():
            result += chr(((ord(char) - 65 + shift) % 26) + 65)
        else:
            result += char
    return result
```

Fungsi caesar_encrypt:

- Menerima dua parameter: text (teks asli) dan shift (jumlah pergeseran).
- text.upper() mengubah teks menjadi huruf besar semua agar konsisten.
- char.isalpha() memastikan hanya huruf alfabet yang dienkripsi.
- ord(char) mengubah huruf menjadi kode ASCII (A=65, B=66, dst).
- Rumus ((ord(char) - 65 + shift) % 26) + 65 menerapkan enkripsi Caesar.
- Hasil akhirnya disimpan dalam variabel result.

```
def caesar_decrypt(text, shift):
    result = ""
    for char in text.upper():
        if char.isalpha():
            result += chr(((ord(char) - 65 - shift) % 26) + 65)
        else:
            result += char
    return result
```

Fungsi caesar_decrypt:

- Melakukan proses kebalikan (dekripsi) dari caesar_encrypt.
- Menggeser huruf ke arah kiri (dikurangi shift).
- Rumus (ord(char) - 65 - shift) % 26 digunakan untuk mengembalikan huruf asli.

```

plain = input("Masukkan teks asli: ")
shift = int(input("Masukkan jumlah pergeseran (0-25): "))

cipher = caesar_encrypt(plain, shift)
decrypted = caesar_decrypt(cipher, shift)

print("\nCiphertext:", cipher)
print("Hasil Dekripsi:", decrypted)

```

Bagian Input dan Output:

- Pengguna memasukkan teks asli dan nilai pergeseran.
- Program akan menampilkan **Ciphertext** dan **Hasil Dekripsi**.

```

with open("hasil_caesar.txt", "w") as f:
    f.write(f"Teks Asli: {plain}\n")
    f.write(f"Pergeseran: {shift}\n")
    f.write(f"Ciphertext: {cipher}\n")
    f.write(f"Dekripsi: {decrypted}\n")

print("\n✓ Hasil disimpan di file 'hasil_caesar.txt'")

```

Untuk menyimpan hasil proses Caesar Cipher (teks asli, pergeseran, ciphertext, dan dekripsi) ke dalam file hasil_caesar.txt.

Input

```

PS C:\Users\ffffit\OneDrive\Desktop\cipher> python caesar.py
Masukkan teks asli: Hello world
Masukkan jumlah pergeseran (0-25): 5

```

Output

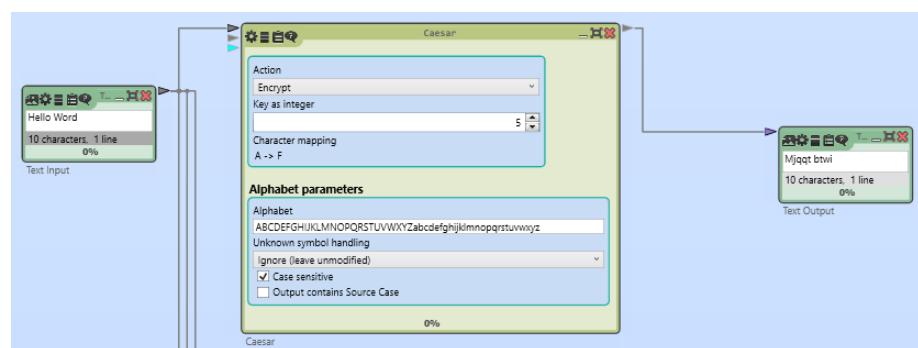
```

Ciphertext: MJQQT BTWQI
Hasil Dekripsi: HELLO WORLD

✓ Hasil disimpan di file 'hasil_caesar.txt'

```

1.4 Implementasi Cryptool



Pengaturan pada gambar

- Action: Encrypt
- Key as integer: 5
- Character mapping: A → F
- Alphabet: A–Z, a–z
- Case sensitive: Dicentang (huruf besar dan kecil diperlakukan berbeda)

Input:

Hello Word

Proses:

Setiap huruf digeser 5 posisi maju di alfabet.

Contoh:

H → M

e → j

l → q

o → t

W → B

o → t

r → w

d → i

Output (ciphertext):

Mjqqt Btwqi

Perubahan terjadi karena setiap huruf plaintext digeser sejauh 5 langkah ke depan dalam alfabet.

Algoritma ini sederhana, namun mudah dipecahkan jika seseorang mengetahui bahwa metode Caesar digunakan.

1.5 Kelemahan Caesar Cipher

- **Ruang kunci sangat kecil**

Hanya ada 26 kemungkinan kunci (0–25), sehingga sangat mudah untuk *di-brute force*.

- **Tidak aman terhadap analisis frekuensi**

Pola huruf pada ciphertext masih mirip dengan plaintext. Misalnya huruf “E” yang sering muncul tetap dapat dikenali setelah pergeseran.

- **Tidak menggunakan konsep kunci rahasia modern**

Semua pihak bisa dengan mudah menebak pergeseran jika tahu satu pasangan huruf.

2. VIGENERE

2.1 Pengertian

Vigenère Cipher adalah pengembangan dari Caesar Cipher yang menggunakan kata kunci (key) untuk menentukan jumlah pergeseran berbeda pada setiap huruf plaintext.

Algoritma ini termasuk cipher polialfabetik, karena tiap huruf dienkripsi dengan alfabet berbeda tergantung huruf kunci.

Penemunya adalah Blaise de Vigenère pada abad ke-16. Cipher ini dahulu dianggap “tidak dapat dipecahkan” selama lebih dari 300 tahun.

2.2 Rumus Matematis

$$E_i = (P_i + K_i) \bmod 26$$

$$D_i = (E_i - K_i) \bmod 26$$

Keterangan:

- P_i = huruf plaintext ke-i
- K_i = huruf kunci ke-i (diulang sesuai panjang teks)
- E_i = huruf ciphertext ke-i

2.3 Implementasi Python

```
def vigenere_encrypt(plaintext, key):
    ciphertext = ""
    key = key.upper()
    i = 0
    for char in plaintext.upper():
        if char.isalpha():
            shift = ord(key[i % len(key)]) - 65
            encrypted_char = chr((ord(char) - 65 + shift) % 26) + 65
            ciphertext += encrypted_char
            i += 1
        else:
            ciphertext += char
    return ciphertext

def vigenere_decrypt(ciphertext, key):
    plaintext = ""
    key = key.upper()
    i = 0
    for char in ciphertext.upper():
        if char.isalpha():
            shift = ord(key[i % len(key)]) - 65
            decrypted_char = chr((ord(char) - 65 - shift) % 26) + 65
            plaintext += decrypted_char
            i += 1
        else:
            plaintext += char
    return plaintext

# Contoh penggunaan
plain = input("Masukkan teks asli: ")
key = input("Masukkan kunci: ")

cipher = vigenere_encrypt(plain, key)
print("\nCiphertext:", cipher)

decrypted = vigenere_decrypt(cipher, key)
print("Hasil Dekripsi:", decrypted)

# Simpan hasil ke file
with open("hasil_vigenere.txt", "w") as f:
    f.write(f"Teks Asli: {plain}\n")
    f.write(f"Kunci: {key}\n")
    f.write(f"Ciphertext: {cipher}\n")
    f.write(f"Hasil Dekripsi: {decrypted}\n")

print("\nHasil disimpan di file 'hasil_vigenere.txt'")
```

Penjelasan code

```
def vigenere_encrypt(plaintext, key):
    ciphertext = ""
    key = key.upper()
    i = 0
    for char in plaintext.upper():
        if char.isalpha():
            shift = ord(key[i % len(key)]) - 65
            encrypted_char = chr((ord(char) - 65 + shift) % 26) + 65
            ciphertext += encrypted_char
            i += 1
        else:
            ciphertext += char
    return ciphertext
```

- `key.upper()` memastikan semua huruf pada kunci adalah huruf besar.
- Program membaca setiap karakter pada plaintext.

- Jika karakter merupakan huruf (char.isalpha()):
- Menghitung pergeseran (shift) berdasarkan huruf pada kunci.
- Melakukan pergeseran huruf sesuai dengan nilai shift.
- Jika bukan huruf (misalnya spasi atau tanda baca), karakter tersebut ditambahkan tanpa perubahan.
- Hasil akhir dikembalikan sebagai ciphertext.

```
def vigenere_decrypt(ciphertext, key):
    plaintext = ""
    key = key.upper()
    i = 0
    for char in ciphertext.upper():
        if char.isalpha():
            shift = ord(key[i % len(key)]) - 65
            decrypted_char = chr(((ord(char) - 65 - shift) % 26) + 65)
            plaintext += decrypted_char
            i += 1
        else:
            plaintext += char
    return plaintext
```

- Sama seperti fungsi enkripsi, tetapi arah pergeseran dibalik.
- `ord(char) - 65 - shift` digunakan untuk mengembalikan huruf ke posisi semula.
- Hasil akhir adalah **plaintext** yang sudah didekripsi.

```
plain = input("Masukkan teks asli: ")
key = input("Masukkan kunci: ")

cipher = vigenere_encrypt(plain, key)
print("\nCiphertext:", cipher)

decrypted = vigenere_decrypt(cipher, key)
print("Hasil Dekripsi:", decrypted)
```

- Pengguna memasukkan teks dan kunci.
- Program mengenkripsi teks menjadi ciphertext.
- Program kemudian mendekripsi kembali ciphertext untuk memastikan hasilnya benar.

```
with open("hasil_vigenere.txt", "w") as f:
    f.write(f"Tekst Asli: {plain}\n")
    f.write(f"Kunci: {key}\n")
    f.write(f"Ciphertext: {cipher}\n")
    f.write(f"Hasil Dekripsi: {decrypted}\n")

print("\nHasil disimpan di file 'hasil_vigenere.txt'")
```

- Membuat file baru bernama hasil_vigenere.txt.
- Menulis hasil input, kunci, ciphertext, dan hasil dekripsi ke dalam file.
- Memberikan notifikasi ke pengguna bahwa hasil telah disimpan.

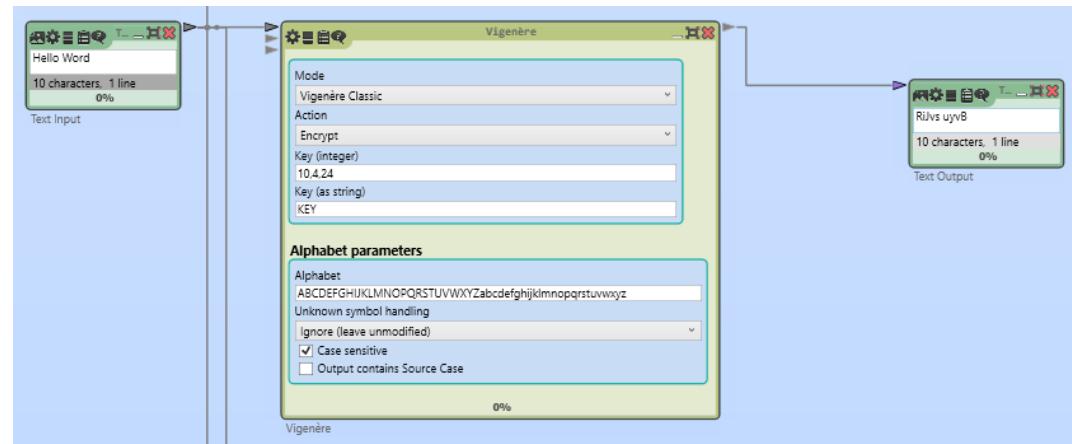
Input

```
Masukkan teks asli: Hello World
Masukkan kunci: key
```

Output

```
Ciphertext: RIJVS UYVJN
Hasil Dekripsi: HELLO WORLD
Hasil disimpan di file 'hasil_vigenere.txt'
```

2.4 Implementasi Cryptool



Pengaturan pada gambar

- Mode: Vigenère Classic
- Action: Encrypt
- Key (string): KEY
- Alphabet: A–Z, a–z
- Case sensitive: Dicentang
- Unknown symbol handling: Ignore

Input:

Hello Word

Proses:

Kunci "KEY" diulang sepanjang teks:

Plaintext : H e l l o W o r d

Key : K E Y K E Y K E Y

Setiap huruf pada plaintext digeser sesuai nilai alfabet dari huruf kunci:

- H (shift K=10) → R
- e (shift E=4) → i
- l (shift Y=24) → J
- l (shift K=10) → v
- o (shift E=4) → s
- (spasi tetap)
- W (shift Y=24) → u
- o (shift K=10) → y
- r (shift E=4) → v
- d (shift Y=24) → æ

Output (ciphertext):

RiJvs uyvæ

(Catatan: simbol “æ” muncul karena hasil pergeseran melampaui alfabet ASCII normal.)

Setiap huruf pada teks asli digeser berdasarkan nilai berbeda dari huruf kunci, sehingga hasil enkripsi lebih kompleks dibanding Caesar Cipher.

Vigenère Cipher lebih sulit dipecahkan karena pola pergeseran berubah-ubah mengikuti kunci.

2.5 Analisis Kelemahan

- Jika panjang kunci diketahui, cipher bisa dipecahkan dengan Kasiski Test.
- Tidak cocok untuk pesan panjang karena pola berulang mudah dianalisis.

3. AFFINE

3.1 Pengertian

Affine Cipher adalah bentuk cipher substitusi monoalfabetik yang menggunakan fungsi linear dalam operasi modulo. Cipher ini bekerja dengan mengonversi huruf menjadi angka ($A=0, \dots, Z=25$), lalu mengenkripsinya menggunakan rumus matematika modular.

Cipher ini memperkenalkan konsep aritmatika modular ke dalam kriptografi klasik.

3.2 Rumus Matematis

$$E(x) = (a \times x + b) \bmod 26$$

$$D(x) = a^{-1} \times (x - b) \bmod 26$$

Keterangan:

- a = konstanta multiplikatif (harus relatif prima terhadap 26)
- b = konstanta aditif (offset)
- a^{-1} = invers modulo dari a terhadap 26

3.3 Implementasi Python

```

def mod_inverse(a, m):
    for i in range(1, m):
        if (a * i) % m == 1:
            return i
    return None

def affine_encrypt(plaintext, a, b):
    ciphertext = ""
    for char in plaintext.upper():
        if char.isalpha():
            x = ord(char) - 65
            y = (a * x + b) % 26
            ciphertext += chr(y + 65)
        else:
            ciphertext += char
    return ciphertext

def affine_decrypt(ciphertext, a, b):
    plaintext = ""
    a_inv = mod_inverse(a, 26)
    if a_inv is None:
        raise ValueError("a tidak memiliki invers modulo 26.")
    for char in ciphertext.upper():
        if char.isalpha():
            y = ord(char) - 65
            x = (a_inv * (y - b)) % 26
            plaintext += chr(x + 65)
        else:
            plaintext += char
    return plaintext

plain = input("Masukkan teks asli: ")
a = int(input("Masukkan nilai a (harus relatif prima dengan 26): "))
b = int(input("Masukkan nilai b: "))

cipher = affine_encrypt(plain, a, b)
decrypted = affine_decrypt(cipher, a, b)

print("\nCiphertext:", cipher)
print("Hasil Dekripsi:", decrypted)

with open("hasil_affine.txt", "w") as f:
    f.write(f"teks Asli: {plain}\n")
    f.write(f"a: {a}, b: {b}\n")
    f.write(f"Ciphertext: {cipher}\n")
    f.write(f"Dekripsi: {decrypted}\n")

print("\n✓ Hasil disimpan di file 'hasil_affine.txt'")

```

Penjelasan Code

```

def mod_inverse(a, m):
    for i in range(1, m):
        if (a * i) % m == 1:
            return i
    return None

```

Fungsi ini mencari invers modulo dari a terhadap m.

Artinya, mencari bilangan i sehingga $(a * i) \% m == 1$.

- Contoh: jika $a = 5$ dan $m = 26$, maka invers dari $5 \text{ mod } 26$ adalah 21 karena $(5 * 21) \% 26 = 1$.
- Fungsi ini penting karena dibutuhkan saat proses dekripsi pada Affine Cipher.

```

def affine_encrypt(plaintext, a, b):
    ciphertext = ""
    for char in plaintext.upper():
        if char.isalpha():
            x = ord(char) - 65
            y = (a * x + b) % 26
            ciphertext += chr(y + 65)
        else:
            ciphertext += char
    return ciphertext

```

Fungsi untuk enkripsi (Affine Cipher).

- Rumus enkripsi:

$$E(x) = (a \times x + b) \bmod 26$$

- x adalah nilai huruf (A=0, B=1, dst.)
- a dan b adalah kunci.
- $\text{ord}(\text{char}) - 65$ mengubah huruf menjadi angka 0–25.
- Hasil y dikonversi kembali ke huruf dengan $\text{chr}(y + 65)$.

Jika karakter bukan huruf (`isalpha()` = False), karakter tersebut tidak diubah (misalnya spasi atau tanda baca).

```
def affine_decrypt(ciphertext, a, b):
    plaintext = ""
    a_inv = mod_inverse(a, 26)
    if a_inv is None:
        raise ValueError("a tidak memiliki invers modulo 26.")
    for char in ciphertext.upper():
        if char.isalpha():
            y = ord(char) - 65
            x = (a_inv * (y - b)) % 26
            plaintext += chr(x + 65)
        else:
            plaintext += char
    return plaintext
```

Fungsi untuk dekripsi (Affine Cipher).

- Rumus dekripsi:

$$D(y) = a^{-1} \times (y - b) \bmod 26$$

- `a_inv` adalah invers dari a terhadap 26, diperoleh dari `mod_inverse(a, 26)`.
- Jika a tidak memiliki invers modulo 26 (artinya tidak relatif prima terhadap 26), maka dekripsi gagal.

```
plain = input("Masukkan teks asli: ")
a = int(input("Masukkan nilai a (harus relatif prima dengan 26): "))
b = int(input("Masukkan nilai b: "))

cipher = affine_encrypt(plain, a, b)
decrypted = affine_decrypt(cipher, a, b)
```

Bagian ini:

- Meminta input dari pengguna:
 - Teks asli (plaintext)

o Kunci a dan b

- Melakukan **enkripsi** dan **dekripsi** menggunakan fungsi yang sudah didefinisikan.

```
print("\nCiphertext:", cipher)
print("Hasil Dekripsi:", decrypted)
```

Menampilkan hasil enkripsi dan dekripsi ke layar.

```
with open("hasil_affine.txt", "w") as f:
    f.write(f"Tekst Asli: {plain}\n")
    f.write(f"a: {a}, b: {b}\n")
    f.write(f"Ciphertext: {cipher}\n")
    f.write(f"Dekripsi: {decrypted}\n")

print("\n✓ Hasil disimpan di file 'hasil_affine.txt'")
```

- Menyimpan hasil proses ke file hasil_affine.txt.
- File ini berisi teks asli, nilai kunci (a, b), hasil enkripsi, dan hasil dekripsi.
- Menampilkan pesan bahwa hasil telah berhasil disimpan.

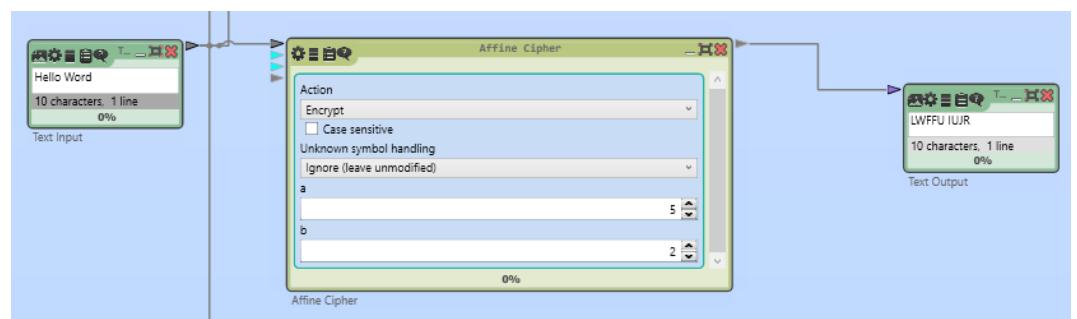
Input

```
PS C:\Users\ffffit\OneDrive\Desktop\cipher> python affine.py
Masukkan teks asli: Hello World
Masukkan nilai a (harus relatif prima dengan 26): 5
Masukkan nilai b: 2
```

Output

```
Ciphertext: LWFFU IUJFR
Hasil Dekripsi: HELLO WORLD
✓ Hasil disimpan di file 'hasil_affine.txt'
```

3.4 Implementasi Cryptool



a. Sumber teks (Text Input)

- Kotak kiri berisi teks asli: "Hello Word" (10 karakter, 1 baris).

- Teks ini akan dikirim ke node Affine Cipher.

b. **Pengaturan node “Affine Cipher” (tengah)**

- **Action:** Encrypt — node melakukan proses *enkripsi* (bukan dekripsi).
- **Case sensitive:** (terlihat kotak ada, kemungkinan *tidak dicentang* pada gambar) — bila dicentang, huruf besar/kecil dibedakan; bila tidak, biasanya semua diubah ke huruf besar sebelum diproses.
- **Unknown symbol handling:** Ignore (leave unmodified) — karakter yang bukan huruf (mis. spasi, angka, simbol) tidak diubah dan dibiarkan sama di output.
- **a = 5, b = 2** — kedua nilai kunci yang dipakai dalam rumus Affine.

c. **Rumus Affine yang dipakai**

- Enkripsi per-huruf ($A=0, B=1, \dots, Z=25$):
 $y = (a * x + b) \text{ mod } 26$
di mana x = nilai numerik huruf plaintext, y = nilai huruf ciphertext.
- Penting: a harus relatif prima terhadap 26 (untuk dekripsi memungkinkan).

d. **Contoh per-huruf (menghasilkan output yang terlihat)**

Plaintext (uppercase): H E L L O _ W O R D

Nilai x ($A=0$): 7 4 11 11 14 _ 22 14 17 3

Hitung $y = (5*x + 2) \text{ mod } 26$:

- H: $(5*7 + 2) = 37 \rightarrow 37 \text{ mod } 26 = 11 \rightarrow L$
- E: $(5*4 + 2) = 22 \rightarrow W$
- L: $(5*11 + 2) = 57 \rightarrow 57 \text{ mod } 26 = 5 \rightarrow F$
- L → F
- O: $(5*14 + 2) = 72 \rightarrow 72 \text{ mod } 26 = 20 \rightarrow U$
- spasi → tetap spasi (Ignore)
- W: $(5*22 + 2) = 112 \rightarrow 112 \text{ mod } 26 = 8 \rightarrow I$
- O → U
- R: $(5*17 + 2) = 87 \rightarrow 87 \text{ mod } 26 = 9 \rightarrow J$

- $D: (5*3 + 2) = 17 \rightarrow R$

Jadi ciphertext: **LWFFU IUJR** (ini sama dengan yang muncul di kotak kanan).

e. Output (Text Output, kanan)

- Menampilkan hasil enkripsi: LWFFU IUJR (10 karakter, 1 line).
- Ini adalah hasil enkripsi sesuai pengaturan ($a=5$, $b=2$) dan aturan pengecualian simbol/spasi.

f. Catatan penting

- Jika kamu ingin **mendekripsi**, pilih Action = Decrypt dan pastikan a memiliki invers modulo 26 (mis. untuk $a=5$, inversnya 21).
- Jika Case sensitive diaktifkan, node harus mendukung mempertahankan kasus (huruf besar/kecil) — jika tidak, biasanya konversi ke uppercase dilakukan sebelum enkripsi.

3.5 Analisis Kelemahan

- Ruang kunci sangat kecil — hanya 312 kombinasi (12×26).
- Monoalfabetik — setiap huruf plaintext selalu dipetakan ke huruf ciphertext yang sama.
- Rentan analisis frekuensi — pola frekuensi huruf tetap, mudah dipecahkan statistik.
- Known-plaintext mudah — dua pasangan plaintext-ciphertext cukup untuk cari a dan b .
- Tidak aman terhadap brute-force — semua kunci dapat dicoba cepat oleh komputer.
- Tidak menangani karakter non-alfabet dengan baik — spasi/angka/simbol biasanya tidak dienkripsi secara berarti.
- Tidak cocok untuk komunikasi modern — tidak memenuhi standar keamanan kriptografi saat ini.
- Mudah dipadankan dengan teknik kripto lainnya — kombinasi atau transformasi sederhana masih rentan; butuh cipher yang jauh lebih kuat (mis. AES).
- Tidak aman terhadap brute-force — semua kunci dapat dicoba cepat oleh komputer.

- Tidak menangani karakter non-alfabet dengan baik — spasi/angka/simbol biasanya tidak dienkripsi secara berarti.
- Tidak cocok untuk komunikasi modern — tidak memenuhi standar keamanan kriptografi saat ini.
- Mudah dipadankan dengan teknik kripto lainnya — kombinasi atau transformasi sederhana masih rentan; butuh cipher yang jauh lebih kuat (mis. AES).

4. PLAYFAIR

4.1 Pengertian

Playfair Cipher adalah algoritma enkripsi berbasis digraf (pasangan huruf) yang diperkenalkan oleh Charles Wheatstone dan dipopulerkan oleh Lord Playfair pada tahun 1854.

Cipher ini menggunakan tabel 5x5 yang diisi dengan huruf-huruf alfabet berdasarkan kata kunci. Huruf “J” biasanya digabung dengan “I” agar jumlah total tetap 25 huruf.

Pesan diubah menjadi pasangan huruf (contoh: “HELLO” → “HE LX LO”), lalu setiap pasangan dienkripsi dengan aturan:

- Jika dua huruf berada di baris yang sama, maka ambil huruf di sebelah kanan masing-masing.
- Jika dua huruf berada di kolom yang sama, ambil huruf di bawah masing-masing.
- Jika membentuk persegi panjang, maka tukar kolomnya.

4.2 Rumus / Langkah Enkripsi

1. Bentuk matriks 5x5 dari kata kunci.
2. Hilangkan huruf duplikat.
3. Ganti huruf J dengan I.
4. Pisahkan teks menjadi pasangan huruf.
5. Terapkan aturan substitusi sesuai posisi huruf.

4.3 Implementasi Python

```

def generate_table(key):
    alphabet = 'ABCDEFGHIJKLMNPQRSTUVWXYZ' # Tanpa J
    table = ''
    for c in key.upper() + alphabet:
        if c not in table:
            table += c
    return [table[i:i+5] for i in range(0, 25, 5)]

def find_position(matrix, char):
    for r in range(5):
        for c in range(5):
            if matrix[r][c] == char:
                return r, c

def playfair_encrypt(plaintext, matrix):
    plaintext = plaintext.upper().replace('J', 'I')
    plaintext = ''.join(c for c in plaintext if c.isalpha()))
    i = 0
    pairs = []
    while i < len(plaintext):
        a = plaintext[i]
        b = plaintext[i + 1] if i + 1 < len(plaintext) else 'X'
        if a == b:
            pairs.append(a + 'X')
            i += 1
        else:
            pairs.append(a + b)
            i += 2
    ciphertext = ''
    for pair in pairs:
        a, b = pair[0], pair[1]
        ra, ca = find_position(matrix, a)
        rb, cb = find_position(matrix, b)

        if ra == rb: # baris sama
            ciphertext += matrix[ra][(ca + 1) % 5]
            ciphertext += matrix[rb][(cb + 1) % 5]
        elif ca == cb: # kolom sama
            ciphertext += matrix[(ra + 1) % 5][ca]
            ciphertext += matrix[(rb + 1) % 5][cb]
        else: # persegi
            ciphertext += matrix[ra][cb]
            ciphertext += matrix[rb][ca]
    return ciphertext

def playfair_decrypt(ciphertext, matrix):
    plaintext = ''
    for i in range(0, len(ciphertext), 2):
        a, b = ciphertext[i], ciphertext[i + 1]
        ra, ca = find_position(matrix, a)
        rb, cb = find_position(matrix, b)

        if ra == rb: # baris sama
            plaintext += matrix[ra][(ca - 1) % 5]
            plaintext += matrix[rb][(cb - 1) % 5]
        elif ca == cb: # kolom sama
            plaintext += matrix[(ra - 1) % 5][ca]
            plaintext += matrix[(rb - 1) % 5][cb]
        else: # persegi
            plaintext += matrix[ra][cb]
            plaintext += matrix[rb][ca]
    return plaintext

# ***** Main Program *****
key = input("Masukkan kunci Playfair: ")
plain = input("Masukkan teks asli: ")

matrix = generate_table(key)
print("\nMatriks Playfair:")
for row in matrix:
    print(row)

cipher = playfair_encrypt(plain, matrix)
print("\nCiphertext:", cipher)

decrypted = playfair_decrypt(cipher, matrix)
print("Hasil Dekripsi:", decrypted)

```

Penjelasan Code

```

def generate_table(key):
    alphabet = 'ABCDEFGHIJKLMNPQRSTUVWXYZ' # Tanpa J
    table = ''
    for c in key.upper() + alphabet:
        if c not in table:
            table += c
    return [table[i:i+5] for i in range(0, 25, 5)]

```

Fungsi:

Membuat matriks 5×5 untuk Playfair Cipher berdasarkan *key* yang diberikan.

Penjelasan langkah:

1. Gunakan alfabet A–Z (tanpa J, karena J digabung dengan I).
2. Kunci (key) dijadikan huruf besar dan J diganti I.
3. Gabungkan key + sisa alfabet, tapi hindari huruf ganda.
4. Hasil disusun menjadi list berisi 5 string (5 huruf tiap baris) → matriks 5×5 .

Contoh:

Jika key = "TARQ", maka tabel bisa jadi seperti:

```
[['T', 'A', 'R', 'Q', 'B'],
```

```
['C', 'D', 'E', 'F', 'G'],
```

```
['H', 'I', 'K', 'L', 'M'],
```

```
['N', 'O', 'P', 'S', 'U'],
```

```
['V', 'W', 'X', 'Y', 'Z']]
```

```
def find_position(matrix, char):
    for r in range(5):
        for c in range(5):
            if matrix[r][c] == char:
                return r, c
```

Fungsi:

Mencari **koordinat (baris, kolom)** dari huruf tertentu dalam tabel Playfair.

Contoh:

Jika matrix seperti di atas dan char = 'L', maka hasilnya (2, 3).

```
def playfair_encrypt(plaintext, matrix):
    plaintext = plaintext.upper().replace("J", "I")
    plaintext = ''.join([c for c in plaintext if c.isalpha()])

    i = 0
    pairs = []
    while i < len(plaintext):
        a = plaintext[i]
        b = plaintext[i + 1] if i + 1 < len(plaintext) else 'X'
        if a == b:
            pairs.append(a + 'X')
            i += 1
        else:
            pairs.append(a + b)
            i += 2

    ciphertext = ""
    for pair in pairs:
        a, b = pair[0], pair[1]
        ra, ca = find_position(matrix, a)
        rb, cb = find_position(matrix, b)

        if ra == rb: # baris sama
            ciphertext += matrix[ra][(ca + 1) % 5]
            ciphertext += matrix[rb][(cb + 1) % 5]
        elif ca == cb: # kolom sama
            ciphertext += matrix[(ra + 1) % 5][ca]
            ciphertext += matrix[(rb + 1) % 5][cb]
        else: # persegi
            ciphertext += matrix[ra][cb]
            ciphertext += matrix[rb][ca]
    return ciphertext
```

Fungsi:

Melakukan enkripsi teks dengan aturan Playfair Cipher.

Tahapan:

1. Ubah plaintext jadi huruf besar dan hapus karakter non-huruf.
2. Ganti J jadi I.
3. Pisahkan teks menjadi pasangan huruf (bigram):
 - Jika dua huruf sama, sisipkan 'X' di antara mereka.
 - Jika jumlah huruf ganjil, tambahkan 'X' di akhir.

Contoh:

HELLO → HE LX LO

Untuk setiap pasangan (a, b):

Jika dalam baris yang sama: ganti masing-masing huruf dengan huruf di kanan (loop ke kiri bila di ujung).

Jika dalam kolom yang sama: ganti masing-masing huruf dengan huruf di bawahnya (loop ke atas bila di bawah).

Jika dalam posisi persegi (rectangle): tukar kolom huruf (ambil huruf di sudut yang sama baris tapi kolom lawan).

Contoh aturan rectangle:

A B

C D

→ menghasilkan: B C

Gabungkan semua pasangan untuk menghasilkan **ciphertext**.

```
def playfair_decrypt(ciphertext, matrix):  
    plaintext = ""  
    for i in range(0, len(ciphertext), 2):  
        a, b = ciphertext[i], ciphertext[i + 1]  
        ra, ca = find_position(matrix, a)  
        rb, cb = find_position(matrix, b)  
  
        if ra == rb: # baris sama  
            plaintext += matrix[ra][(ca - 1) % 5]  
            plaintext += matrix[rb][(cb - 1) % 5]  
        elif ca == cb: # kolom sama  
            plaintext += matrix[(ra - 1) % 5][ca]  
            plaintext += matrix[(rb - 1) % 5][cb]  
        else: # persegi  
            plaintext += matrix[ra][cb]  
            plaintext += matrix[rb][ca]  
  
    return plaintext
```

Fungsi:

Mendekripsi ciphertext dengan aturan kebalikan enkripsi.

Aturan kebalikannya:

1. Sama baris: ambil huruf di kiri masing-masing.
2. Sama kolom: ambil huruf di atasnya.
3. Rectangle: tukar kolom seperti pada enkripsi.

Hasil akhir adalah plaintext (kadang masih mengandung huruf tambahan ‘X’ jika disisipkan saat enkripsi).

```
key = input("Masukkan kunci Playfair: ")
plain = input("Masukkan teks asli: ")

matrix = generate_table(key)
print("\nTabel Playfair:")
for row in matrix:
    print(row)

cipher = playfair_encrypt(plain, matrix)
print("\nCiphertext:", cipher)

decrypted = playfair_decrypt(cipher, matrix)
print("Hasil Dekripsi:", decrypted)
```

Fungsi:

- Meminta input kunci dan teks asli dari pengguna.
- Membuat matriks Playfair dari kunci.
- Mengenkripsi dan mendekripsi teks, lalu menampilkan hasilnya.

Input

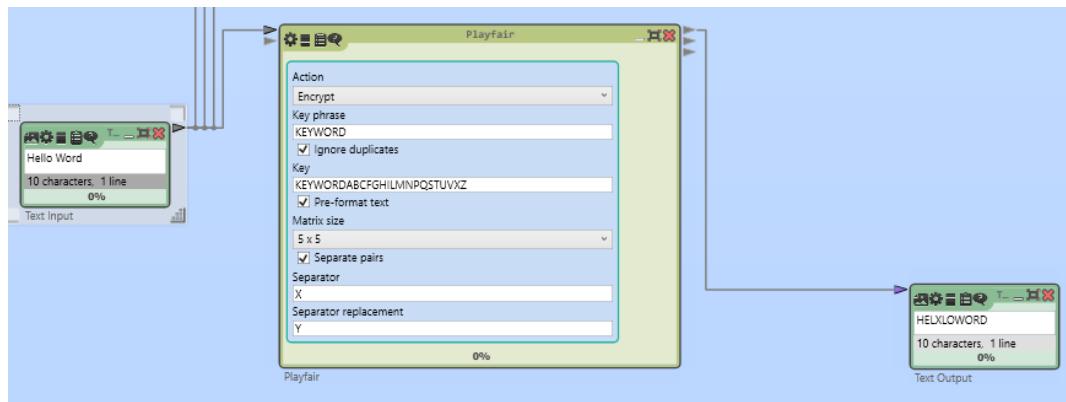
```
PS C:\Users\ffffit\OneDrive\Desktop\cipher> python playfair.py
Masukkan kunci Playfair: KEYWORD
Masukkan teks asli: Hello Word

Tabel Playfair:
KEYMO
RDIABC
FGHIL
MNQPS
TUVWXZ
```

Output

```
Ciphertext: GYIZSCOKDA
Hasil Dekripsi: HELLOWORD
```

4.4 Implementasi Cryptool



1. Text Input

Kotak di sebelah kiri berisi:

Hello Word

Ini adalah teks asli (plaintext) yang akan dienkripsi.

2. Playfair (Modul utama)

Kotak tengah adalah **modul Playfair Cipher**, dengan beberapa pengaturan penting:

Opsi	Nilai	Penjelasan
Action	Encrypt	Menandakan proses yang dilakukan adalah enkripsi (bukan dekripsi).
Key phrase	KEYWORD	Kata kunci yang digunakan untuk membentuk matriks Playfair.
Ignore duplicates	<input checked="" type="checkbox"/> (centang)	Menghapus huruf yang

		berulang dalam kunci.
Key	KEYWORDABCFGHILMNOPQRSTUVWXYZ	Ini hasil dari penghapusan duplikat huruf dari kata kunci, lalu diikuti sisa alfabet (J biasanya digabung dengan I).
Matrix size	5×5	Matriks standar Playfair (5 kolom × 5 baris).
Separate pairs	✓	Teks akan dipisah menjadi pasangan huruf (bigram).
Separator	X	Huruf X digunakan untuk memisahkan huruf yang sama dalam pasangan (contoh: “HELLO” → “HE LX LO”).
Separator replacement	Y	Jika X sudah digunakan,

		maka Y bisa dipakai sebagai alternatif pemisah.
--	--	---

3. Text Output

Kotak di sebelah kanan menampilkan hasil:

HELXLOWORD

Ini adalah ciphertext hasil enkripsi dengan algoritma Playfair berdasarkan kunci "KEYWORD".

4. Cara kerja singkat Playfair Cipher

1. Buat matriks 5×5 berdasarkan kunci "KEYWORD" (tanpa huruf duplikat, J = I).
2. Bagi teks menjadi pasangan huruf. Jika ada huruf ganda dalam satu pasangan, tambahkan X di antaranya.

"HELLO WORD" → "HE LX LO WO RD"

3. Enkripsi setiap pasangan menggunakan aturan Playfair:

Jika huruf berada di baris yang sama, ganti dengan huruf di kanan masing-masing.

Jika di kolom yang sama, ganti dengan huruf di bawah masing-masing.

Jika berbentuk persegi panjang, tukar kolom antar huruf.

Hasil akhirnya adalah **HELXLOWORD**.

Kesimpulan

Gambar tersebut menunjukkan proses enkripsi teks "Hello Word" menggunakan Playfair Cipher dengan kunci "KEYWORD", menghasilkan ciphertext "HELXLOWORD".

Setiap pengaturan dalam modul menentukan bagaimana teks diolah dan huruf tambahan (“X”) dimasukkan untuk menjaga aturan Playfair.

4.5 Analisis Kelemahan

- Rentan terhadap analisis digraf frekuensi.
- Tidak mendukung karakter tunggal atau huruf ganda secara langsung.
- Pola huruf berulang masih bisa dikenali.

5. HILL

5.1 Pengertian

Hill Cipher merupakan cipher berbasis matriks linear algebra, diciptakan oleh Lester S. Hill pada tahun 1929. Cipher ini mengenkripsi blok huruf (biasanya 2 atau 3 sekaligus) menggunakan operasi perkalian matriks dengan kunci, kemudian hasilnya diambil modulo 26.

Hill Cipher merupakan cipher klasik pertama yang benar-benar matematis dan menjadi dasar bagi sistem enkripsi modern berbasis linear transformasi.

5.2 Rumus Matematis

$$C = (K \times P) \bmod 26$$

$$P = (K^{-1} \times C) \bmod 26$$

Keterangan:

- C = ciphertext (vektor hasil enkripsi)
- P = plaintext (vektor input)
- K = matriks kunci (harus memiliki determinan yang memiliki invers mod 26)
- K^{-1} = invers dari matriks kunci mod 26

5.3 Implementasi Python

```

import numpy as np

def hill_encrypt(plaintext, matrix):
    plaintext = plaintext.upper().replace(" ", "")
    while len(plaintext) % 2 != 0:
        plaintext += "X"
    ciphertext = ""
    for i in range(0, len(plaintext), 2):
        pair = [ord(plaintext[i]) - 65, ord(plaintext[i+1]) - 65]
        result = np.dot(matrix, pair) % 26
        ciphertext += chr(result[0] + 65) + chr(result[1] + 65)
    return ciphertext

matrix = np.array([[3, 3],
                  [2, 5]]) # ✅ sama seperti di CrypTool
text = input("Masukkan teks: ")
print("Ciphertext:", hill_encrypt(text, matrix))

```

Penjelasan langkah-langkahnya

1. Input teks → huruf besar semua, spasi dihapus.
2. Padding “X” → jika jumlah huruf ganjil.
3. Konversi huruf ke angka dengan ord() dan dikurangi 65 (karena A = 65 di ASCII).
4. Perhitungan utama:
Mengalikan matriks kunci dengan vektor pasangan huruf, lalu diambil hasilnya mod 26.
5. Konversi kembali ke huruf (chr(result + 65)).
6. Hasilnya adalah ciphertext yang sama seperti di CrypTool.

Kesimpulan

- Kedua gambar menggambarkan algoritma Hill Cipher dengan kunci matriks

$$\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$$

- Input plaintext: HELLOWORLD
- Hasil enkripsi: HIOZEIQJOL
- Implementasi di CrypTool dan Python menghasilkan hasil yang sama, membuktikan bahwa kode Python sudah benar mengikuti logika matematis Hill Cipher.

Input

```

PS C:\Users\ffffit\OneDrive\Desktop\cipher> python hill.py
Masukkan teks: Helloward

```

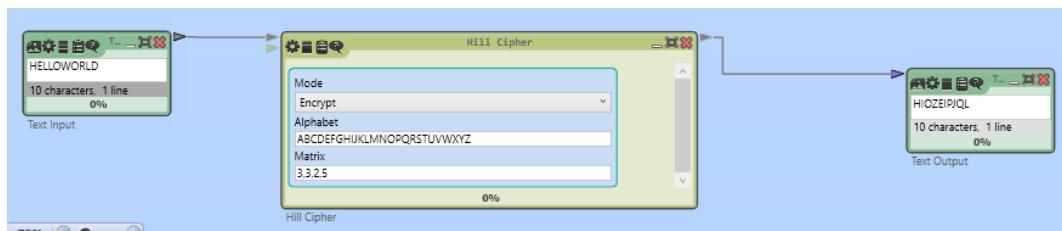
Output

```

Ciphertext: HIOZEIPJAR

```

5.4 Implementasi Cryptoll



Gambar pertama menunjukkan blok Hill Cipher dengan pengaturan berikut:

Parameter	Nilai	Penjelasan
Mode	Encrypt	Artinya proses yang dilakukan adalah enkripsi.
Alphabet	A–Z	Hanya huruf kapital A–Z yang digunakan, tanpa spasi atau simbol.
Matrix	3, 3, 2, 5	Ini adalah kunci matriks dalam bentuk 2×2 : [[3, 3], [2, 5]]
Input text	HELLOWORLD	Plaintext yang akan dienkripsi.
Output text	HIOZEIQJOL	Ciphertext hasil enkripsi.

Cara kerja Hill Cipher

Hill Cipher adalah cipher polialfabetik berbasis aljabar linear.

Langkah umumnya:

1. Konversi huruf ke angka

$$A=0, B=1, \dots, Z=25$$

Contoh: HELLO → [7, 4, 11, 11, 14]

2. Bagi plaintext jadi pasangan (2 huruf)

Karena matriksnya 2×2 , maka teks diproses per dua huruf:

HE, LL, OW, OR, LD

3. Kali matriks kunci dengan tiap pasangan (mod 26)

Gunakan kunci:

$$K = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}$$

Misal untuk pasangan “HE” → [7, 4]:

$$\begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 7 \\ 4 \end{bmatrix} = \begin{bmatrix} 33 \\ 34 \end{bmatrix} \text{mod } 26 = \begin{bmatrix} 7 \\ 8 \end{bmatrix} = "HI"$$

Proses ini diulang untuk semua pasangan.

4. Gabungkan hasil tiap pasangan → Ciphertext

Hasil akhirnya: HIOZEIQJOL.

5.5 Analisis Kelemahan

- Tidak semua matriks dapat digunakan (harus invertible mod 26).
- Jika diketahui pasangan plaintext–ciphertext, kunci dapat dihitung balik.
- Tidak cocok untuk penggunaan modern karena blok kecil mudah dianalisis.

6. PERBANDINGAN ANTAR ALGORITMA (Caesar, Vigenere, Affine, Playfair, Hill)

Cipher	Tipe Enkripsi	Kompleksitas	Tingkat Keamanan	Karakteristik
Caesar	Substitusi monoalfabetik	Rendah	Lemah	Pergeseran huruf tetap
Vigenère	Substitusi polialfabetik	Sedang	Cukup kuat	Menggunakan kunci berulang
Affine	Substitusi linear	Sedang	Lemah	Kombinasi fungsi linear
Playfair	Substitusi digraf	Menengah	Cukup kuat	Menggunakan pasangan huruf
Hill	Cipher matriks	Tinggi	Lebih kuat	Menggunakan aljabar linear